Implementing Temporal Integrity Constraints Using an Active DBMS

David Toman Jan Chomicki

Kansas State University

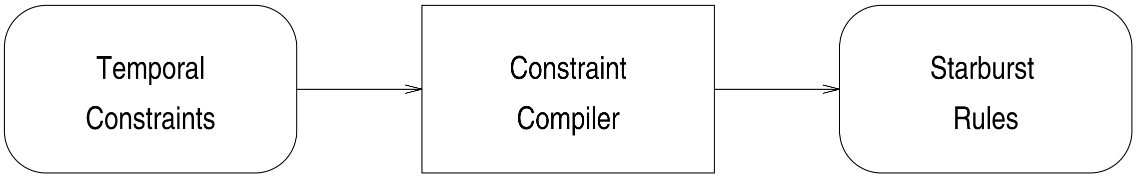
Integrity Constraints

- Declarative vs. Procedural Specification of Constraints
- Static Constraints vs. Temporal (Dynamic) Constraints

Examples:

"Every person has at most two parents"

"Once student drops out, she should not be readmitted"



- Declarative Specification
- No need for additional Run-time System
- Separation of Constraints from the Application
- Historyless constraint checking
- Automatic rejection of non-safe constraints

Specification Language: Past Temporal Logic

Syntax:

1. $R(x_1, \ldots, x_n)$ for all relations R in the database schema
2. $x \rho y$ where $\rho \in \{=, <, \leq, \neq, \geq, >\}$
3. $A \vee B$, $A \wedge B$, $\neg A$, $\exists x.A$, A *since* B, $\bullet A$

Semantics:
Defined w.r.t. the sequence $D_0, D_1, \ldots, D_k$ of database states.

1. FOL formulas have the standard interpretation w.r.t. the appropriate state D_i.
2. $\bullet A$ is true in D_i if $i > 0$ and A is true in D_{i-1}
3. A *since* B is true in D_i if B is true in D_k for some $0 \leq k < i$ and A is true in D_j for all $k < j \leq i$.

Syntactic sugar: $\blacklozenge A \equiv (\text{true} \;\; \textit{since}\; A)$ and $\blacksquare A \equiv \neg \blacklozenge \neg A$

Example:

- "Salary of of an employee cannot decrease"

$$\neg(\exists e, s_1, s_2)(\blacklozenge emp(e, s_1) \wedge emp(e, s_2) \wedge s_2 < s_1)$$

- "Once student drops out, she should not be readmitted"

$$\neg(\exists . x)(admitted(x) \wedge \blacklozenge dropout(x))$$

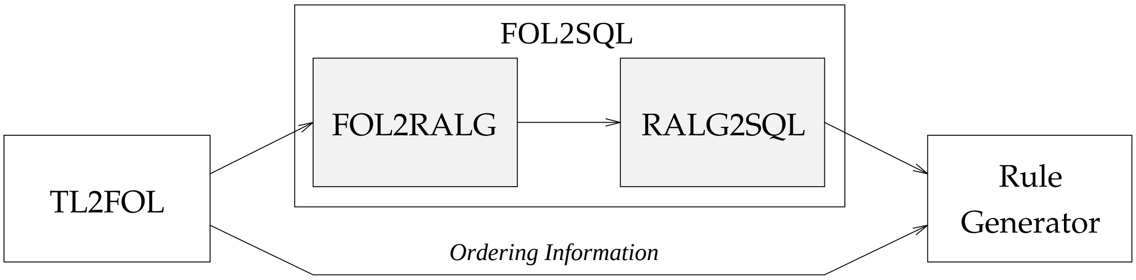
The Target: Starburst Rule System

```
create rule  rule-name
  [when      {inserted(T_i)|deleted(T_i)|updated(T_i)}]
  [when      commit]
  [if        SQL predicate]
   then      SQL actions
  [precedes  list of rule names]
  [follows   list of rule names]
```

- Set oriented vs. Tuple oriented rules
- Access to the previous state of the database (extension of SQL with transition tables)

$$\begin{aligned} r^{n-1} \equiv r^n & - (\texttt{new-updated}(r) \cup \texttt{inserted}(r)) \\ & \cup (\texttt{old-updated}(r) \cup \texttt{deleted}(r)) \end{aligned}$$

Structure of the System



- Modular Design
- The “common” intermediate language of FOL

Temporal Information Management

Each temporal subformula is replaced by an auxiliary relation

$$F \to F[r_\alpha/\alpha] \ \text{ for } \alpha \in \{\bullet A, A \mathit{\ since\ } B\}$$

The auxiliary relations are updated using following definition

α	Auxiliary relation definition
$\bullet A$	$r_\alpha^0 :=$ false; $r_\alpha^n := A^{n-1}$
A *since* B	$r_\alpha^0 :=$ false; $r_\alpha^n := r_\alpha^{n-1} \wedge A^n \vee B^{n-1} \wedge A^n$

- Each auxiliary relation contains references to the current and previous states only.

Rule Generator

- Top-level constraint:

  `create rule` $\mathcal{C}$

`when`	`commit`
`if`	`not exists` $< \mathcal{C} >$
`then`	`rollback work`

- Auxiliary relations:

  `create rule` r_α

`when`	`commit`
`then`	*update* r_α `by` $< \varphi_{r_\alpha} >$
`precedes`	list of rules r_α is used in

- The rule processing is triggered by the transaction commit.
- The order of processing is controlled by the `precedes` keyword.
- The *update* operation is realized by a sequence of insert and delete operations.

Example

Constraint:

"Once student drops out, she should not be readmitted"

Past Temporal Logic formula:

$$\neg(\exists.x)(admitted(x) \wedge \blacklozenge dropout(x))$$

First Order Logic reformulation:

- Top-level constraint $\mathcal{C}$: $\neg(\exists.x)(admitted(x) \wedge r(x))$
- Auxiliary relation r: $r^0(x) :=$ false
  $r^n(x) := r^{n-1}(x) \vee dropout^{n-1}(x)$

Generated Rules:

- Top-level constraint:

```
create rule 𝒞
   when   commit
   if not < ¬(∃.x)(admitted(x) ∧ r(x)) >
   then   rollback work;
```

- Auxiliary relation rematerialization

```
create rule r
   when   commit
   then   insert into r values dropout^{n-1}
   precedes  𝒞;
```

Conclusion

Possible future extensions:

- More powerful Specification Language
- Analysis and Optimization of the Auxiliary Relations
- Optimization of the view rematerialization
- Adopting different Rule system (e.g. tuple based)